CS 3630 Intro to Perception and Robotics

Fitting Models: Hough Transform and RANSAC

Aaron Bobick
School of Interactive Computing

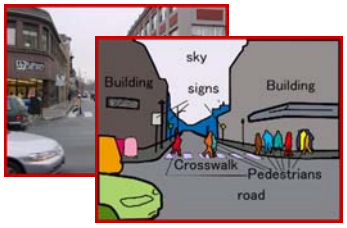
Vision vs Image Processing

- In the first part of compute vision we learn to process images to make new images.

$$F : I(x, y) \longrightarrow I'(x, y)$$

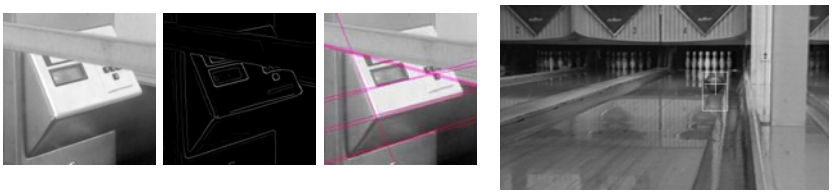
- But real vision:

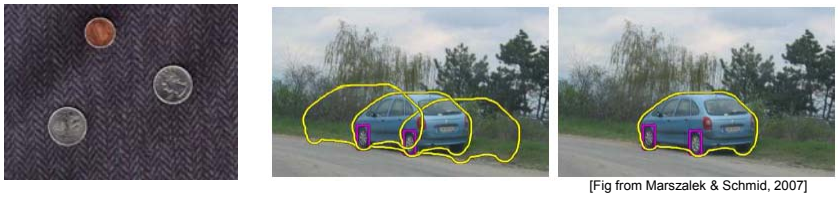
$$F : I(x, y) \longrightarrow \text{good stuff}$$



Fitting a model

- Want to associate a model with observed features





[Fig from Marszalek & Schmid, 2007]

For example, the model could be a line, a circle, or an arbitrary shape.

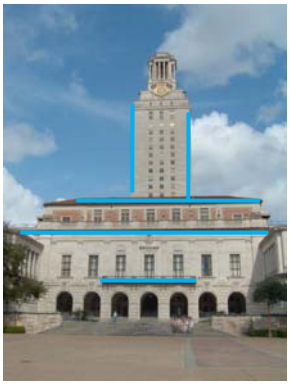
Fitting

- Choose a parametric model to represent a set of features
- Membership criterion is not local
  - Can't tell whether a point belongs to a given model just by looking at that point
- Three main questions:
  - What model represents this set of features best?
  - Which of several model instances gets which feature?
  - How many model instances are there?
- Computational complexity is important
  - It is infeasible to examine every possible set of parameters and every possible combination of features

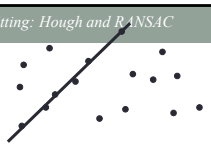
Source: L. Lazebnik

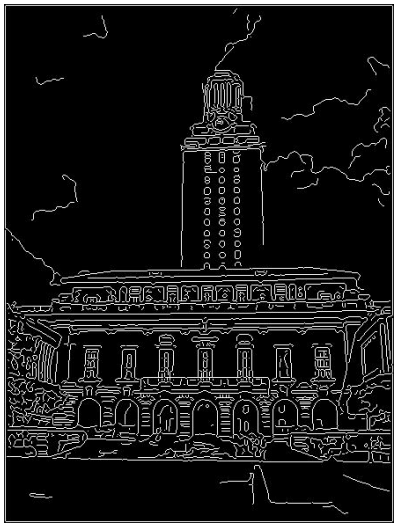
Example: Line fitting

- Why fit lines?
- Many objects characterized by presence of straight lines



Difficulty of line fitting





- **Extra** edge points (clutter), multiple models:
  - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing:**
  - how to find a line that bridges missing evidence?
- **Noise** in measured edge points, orientations:
  - how to detect true underlying parameters?

Fitting: Issues

Case study: Line detection

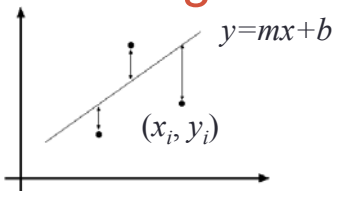
- **Noise** in the measured feature locations
- **Extraneous data:** clutter (outliers), multiple lines
- **Missing data:** occlusions

Slide: S. Lazebnik

Typical least squares line fitting

- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m x_i + b$
- Find (m, b) to minimize

$$E = \sum_{i=1}^{n} (y_i - m x_i - b)^2$$



$$E = \sum_{i=1}^{n} \left(y_i - \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} \right)^2 = \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} \right\|^2 = \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2$$

$\mathbf{b}$

$$= (\mathbf{y} - \mathbf{X}\mathbf{b})^T (\mathbf{y} - \mathbf{X}\mathbf{b}) = \mathbf{y}^T\mathbf{y} - 2(\mathbf{X}\mathbf{b})^T\mathbf{y} + (\mathbf{X}\mathbf{b})^T(\mathbf{X}\mathbf{b})$$

$$\frac{dE}{d\mathbf{b}} = 2\mathbf{X}^T\mathbf{X}\mathbf{b} - 2\mathbf{X}^T\mathbf{y} = 0$$

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y} \Rightarrow \mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

psuedoinverse

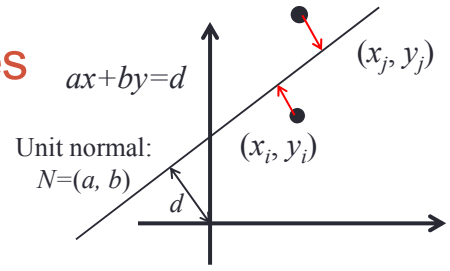
Standard least squares solution to except weird switch of typical names from $\mathbf{A}\mathbf{x}=\mathbf{b}$

Problem with "vertical" least squares

- Not rotation-invariant
- Fails completely for vertical lines

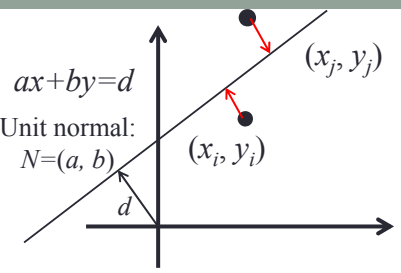
Total least squares

- Distance between point (x_i, y_i) and line $ax+by=d$ where $(a^2+b^2=1)$ is $|ax_i + by_i - d|$
- Find (a, b, d) to minimize the sum of squared perpendicular distances



$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$

Total least squares



- Find (a, b, d) to minimize the sum of squared perpendicular distances

$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$

$$\frac{\partial E}{\partial d} = \sum_{i=1}^{n} -2(ax_i + by_i - d) = 0$$

$$d = \frac{a}{n}\sum_{i=1}^{n} x_i + \frac{b}{n}\sum_{i=1}^{n} x_i = a\bar{x} + b\bar{y}$$

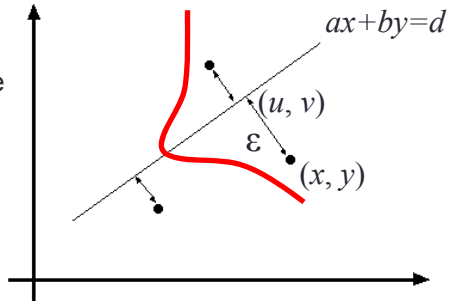
$$E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \left\| \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2 = (UN)^T (UN)$$

$$\frac{dE}{dN} = 2(U^T U)N = 0$$

Solution to $(U^T U)N = 0$, subject to $\|N\|^2 = 1$: eigenvector of $U^T U$ associated with the smallest eigenvalue (Again SVD to least squares solution to *homogeneous linear system* $UN = 0$)

Least squares as likelihood maximization

- **Generative model**: line points are corrupted by Gaussian noise in the direction perpendicular to the line



$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} + \varepsilon \begin{pmatrix} a \\ b \end{pmatrix}$$

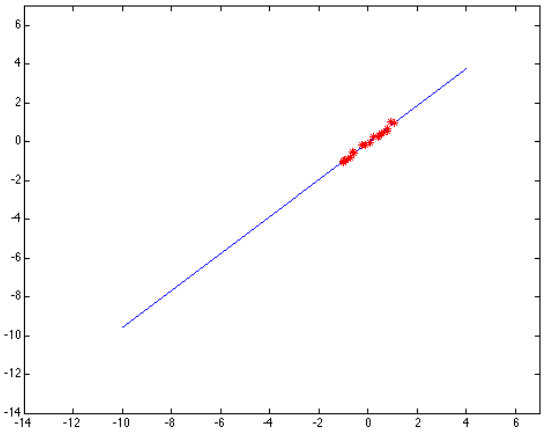
point on the line

noise: sampled from zero-mean Gaussian with std. dev. σ

normal direction

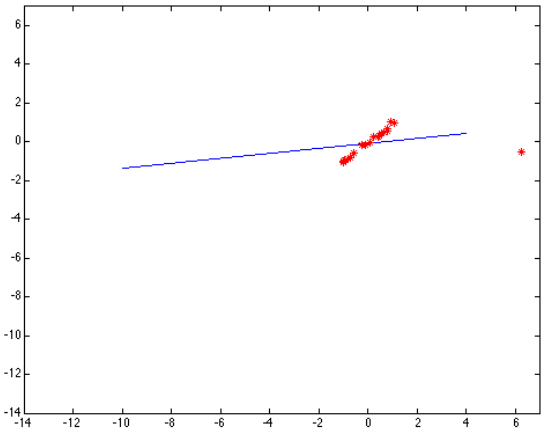
Least squares: Non-robustness to (very) non-Gaussian noise

- Least squares fit to the red points:



Least squares: Non-robustness to (very) non-Gaussian noise

- Least squares fit with an outlier:

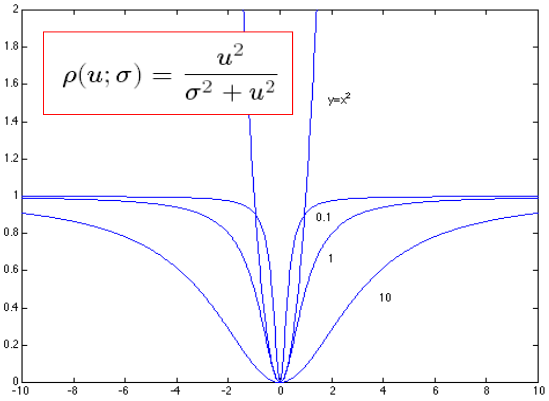


Problem: squared error heavily penalizes outliers

Robust estimators

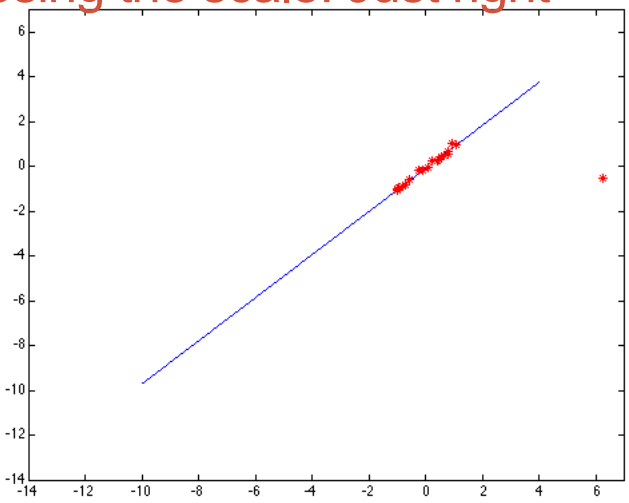
- General approach: minimize $\sum_i \rho(r_i(x_i, \theta); \sigma)$

$r_i(x_i, \theta)$ – residual of ith point w.r.t. model parameters θ
ρ – robust function with scale parameter σ



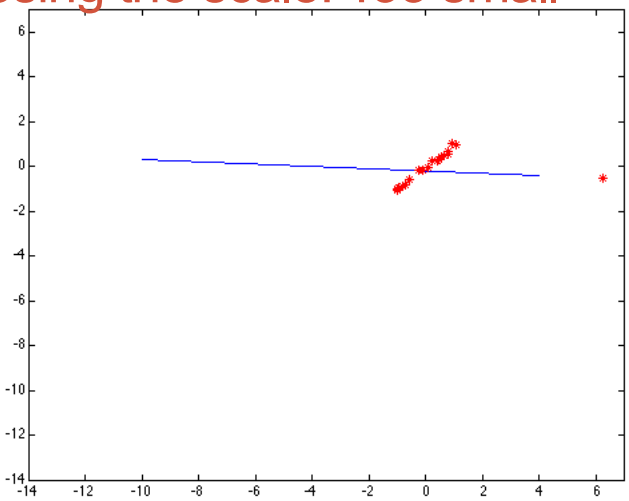
The robust function ρ behaves like squared distance for small values of the residual u but saturates for larger values of u

Choosing the scale: Just right



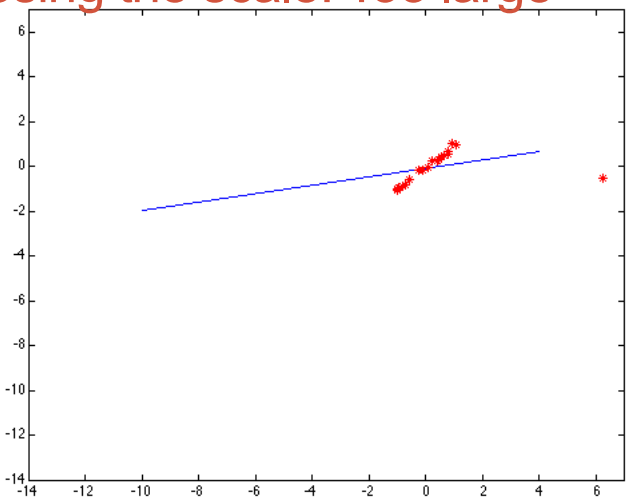
The effect of the outlier is minimized

Choosing the scale: Too small



The error value is almost the same for every point and the fit is very poor

Choosing the scale: Too large



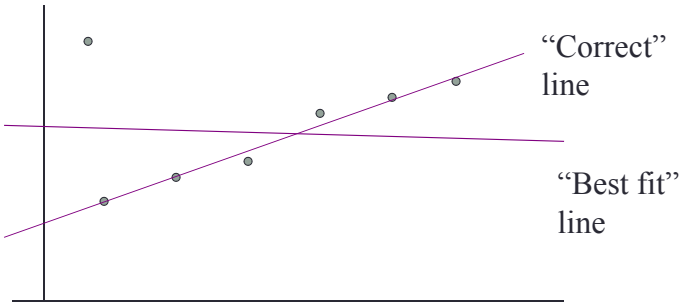
Behaves much the same as least squares

"Find consistent matches"

- Some points (many points) are part of the model we want.
- Some are not
- Need to find the right ones.

Simplest Example

- Fitting a straight line



"Find consistent matches"

- Some points (many points) are part of the model we want.
- Some are not
- Need to find the right ones.

- Two well understood approches:
  - Hough Transform – everyone votes for all the models they are comfortable with. Well supported models are chosen.
  - Random Sample Consensus (RANSAC) – a variety of models are proposed until one is found that is supported by a consensus of voters.

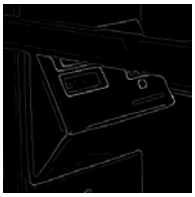
Hough Voting

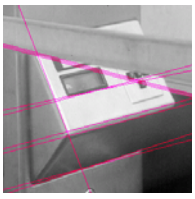
- It's not feasible to check all combinations of features by fitting a model to each possible subset.
- Voting is a general technique where we let the features vote for all models that are compatible with it.
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that receive a lot of votes.
- Noise & clutter features will cast votes too, but typically their votes should be inconsistent with the majority of "good" features.
- Ok if some features not observed, as model can span multiple fragments.

Fitting lines

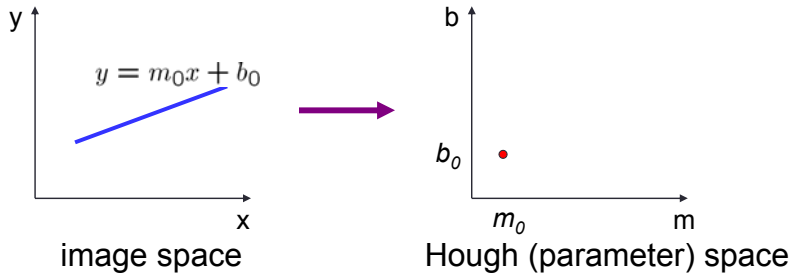
- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?

- **Hough Transform** is a voting technique that can be used to answer all of these
  - Main idea:
  - 1. Record all possible lines on which each edge point lies.
  - 2. Look for lines that get many votes.





Finding lines in an image: Hough space

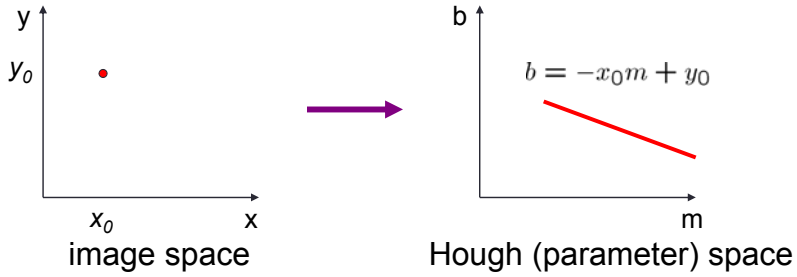


Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that **y = mx + b**

Slide credit: Steve Seitz

Finding lines in an image: Hough space



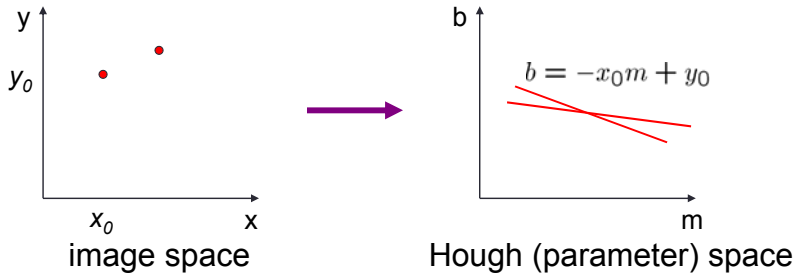
Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that **y = mx + b**
- What does a point (x_0, y_0) in the image space map to?
  - Answer: the solutions of **$b = -x_0 m + y_0$**
  - this is a line in Hough space

Slide credit: Steve Seitz

CS3630 Intro to Perception and Robotics – A. Bobick
Model Fitting: Hough and RANSAC

Finding lines in an image: Hough space



Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that **y = mx + b**
- What does a point (x_0, y_0) in the image space map to?
  - Answer: the solutions of **$b = -x_0 m + y_0$**
  - this is a line in Hough space

Finding lines in an image: Hough *transform*

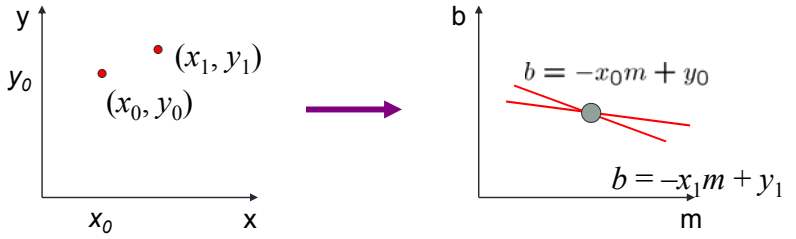


image space Hough (parameter) space

What are the line parameters for the line that contains both (x_0, y_0) and (x_1, y_1)?

- It is the intersection of the lines $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$

Finding lines: Hough algorithm

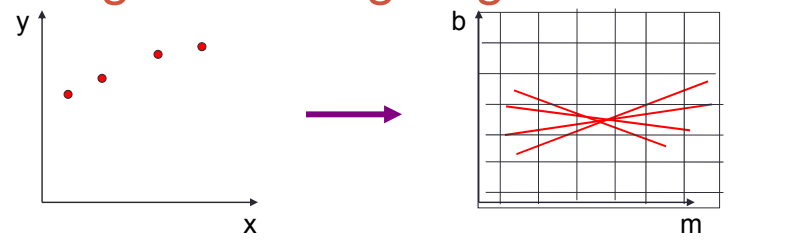
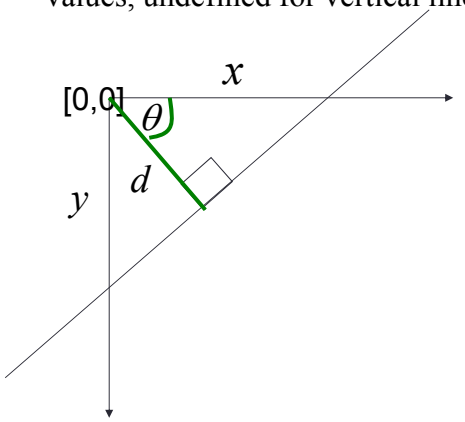


image space Hough (parameter) space

- How can we use this to find the most likely parameters (m,b) for the most prominent line in the image space?
- Let each edge point in image space vote for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

Polar representation for lines

Issues with usual (m,b) parameter space: can take on infinite values, undefined for vertical lines.



d : perpendicular distance from line to origin

θ : angle the perpendicular makes with the x-axis

$$x\cos\theta - y\sin\theta = d$$

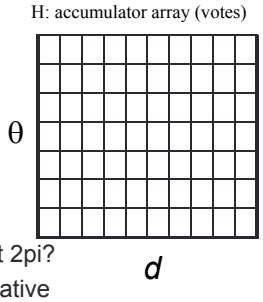
Point in image space → sinusoid segment in Hough space

Hough transform algorithm

Using the polar parameterization:

$$x\cos\theta - y\sin\theta = d$$

H: accumulator array (votes)



Basic Hough transform algorithm

1. Initialize H[d, θ]=0
2. for each edge point I[x,y] in the image
   for θ = 0 to 180 // some quantization; not 2pi?
   $d = x\cos\theta - y\sin\theta$ // maybe negative
   H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x\cos\theta - y\sin\theta$

Space complexity? k^n (n dimensions, k bins each)

Time complexity (in terms of number of voting elements)?

Example: Hough transform for straight lines

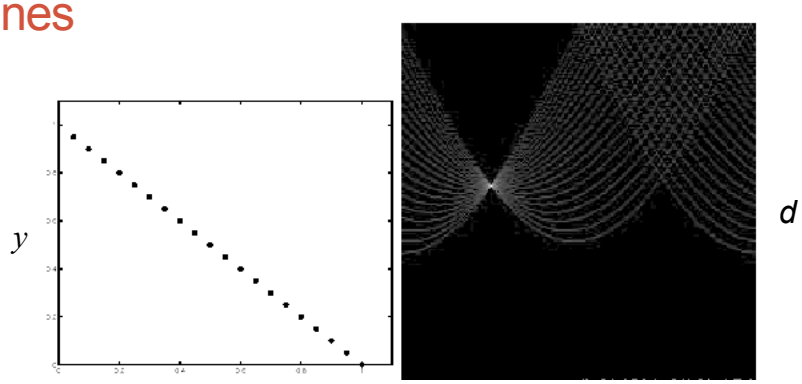
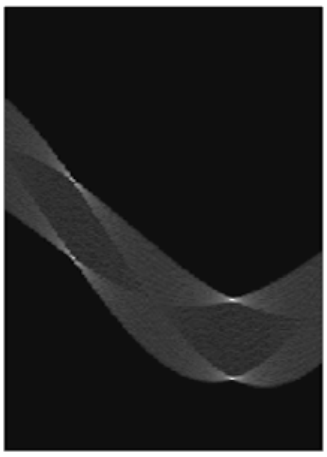


Image space edge coordinates

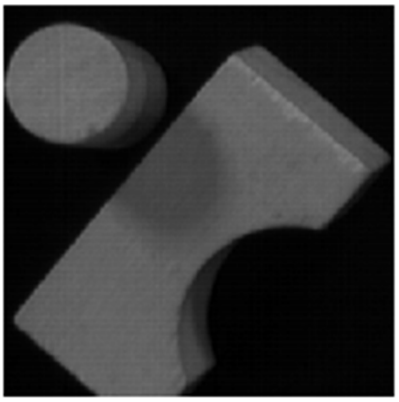
Votes
Bright value = high vote count
Black = no votes

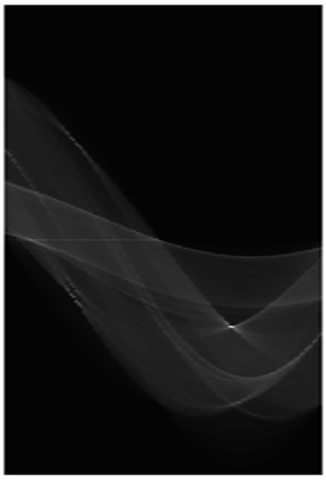
Example: Hough transform for straight lines

Square :

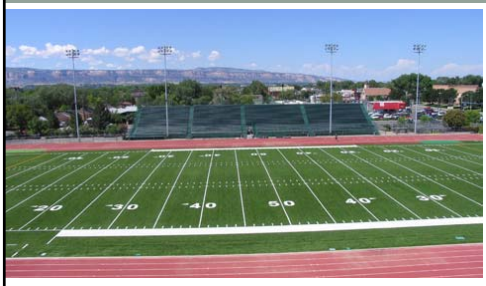


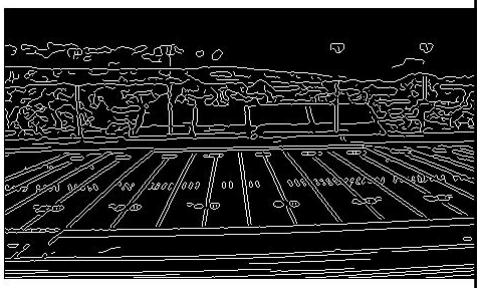
Example: Hough transform for straight lines

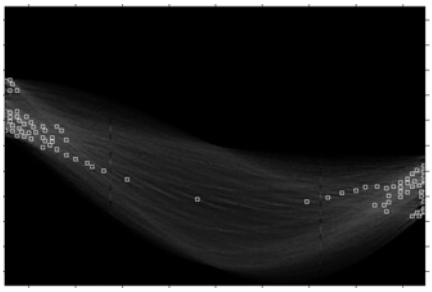


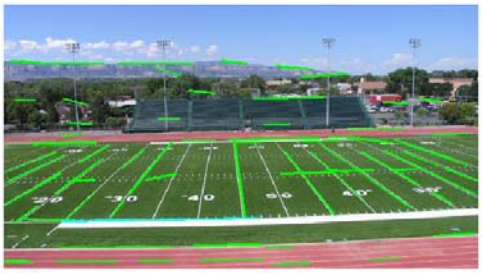


Hough demo..



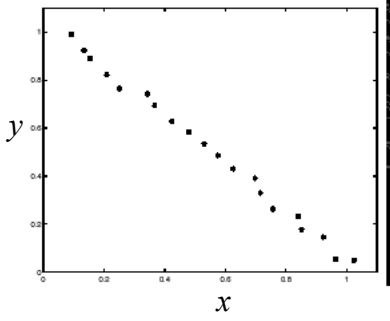






Showing longest segments found

Impact of noise on Hough



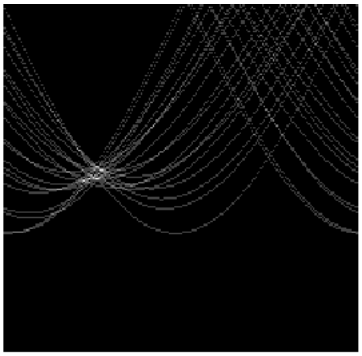


Image space edge coordinates

Votes

What difficulty does this present for an implementation?

Impact of noise on Hough

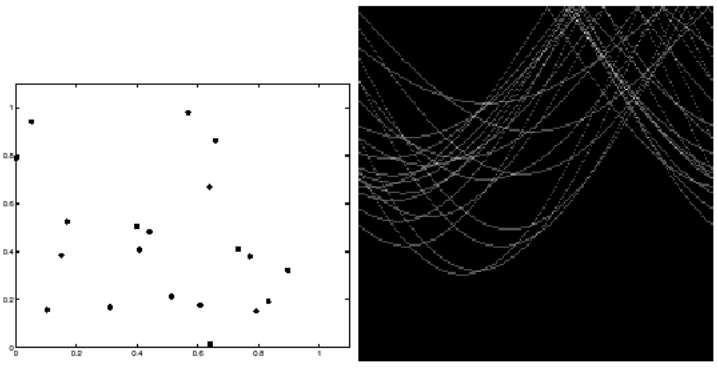


Image space edge coordinates

Votes

Here, everything appears to be "noise", or random edge points, but we still see peaks in the vote space.

Extensions

$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$

$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

- ***Extension 1***: Use the image gradient
  - same
  - for each edge point I[x,y] in the image
    - θ = gradient at (x,y)

      $d = x\cos\theta - y\sin\theta$

    - H[d, θ] += 1
  - same
  - same
- (Reduces degrees of freedom)

- Extension 2
  - give more votes for stronger edges
- Extension 3
  - change the sampling of (d, θ) to give more/less resolution
- Extension 4
  - The same procedure can be used with circles, squares, or any other shape

Hough transform for circles

- Circle: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- For a fixed radius r, unknown gradient direction

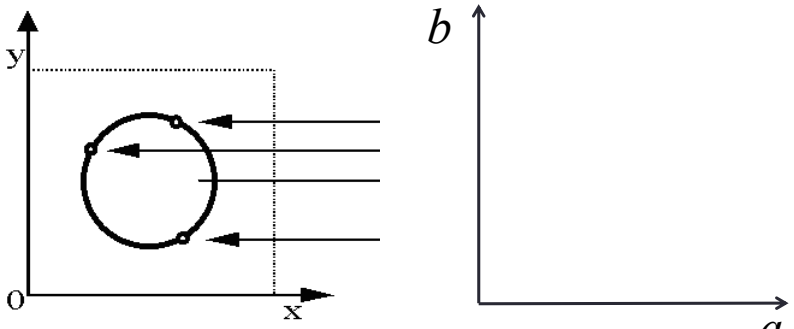


Image space Hough space

Hough transform for circles

- Circle: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- For a fixed radius r, unknown gradient direction

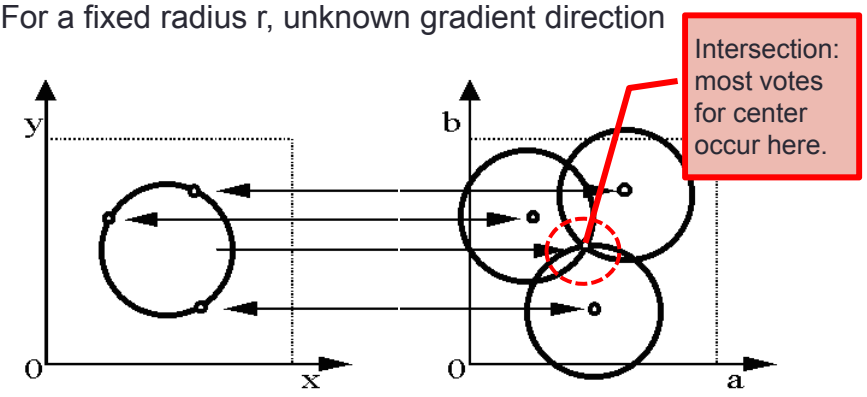


Image space Hough space

Hough transform for circles

- Circle: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- For an unknown radius r, unknown gradient direction

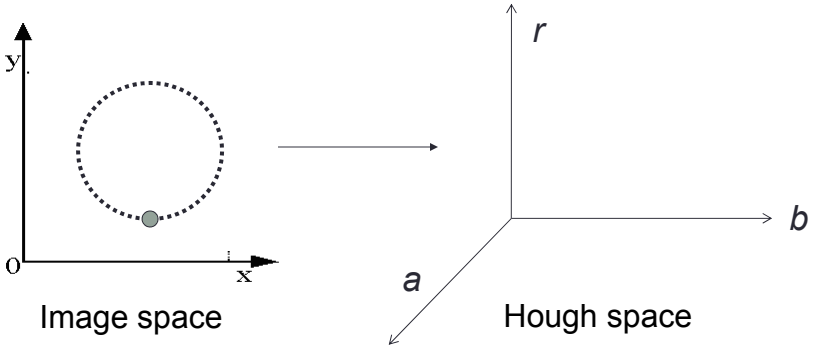


Image space Hough space

CS3630 Intro to Perception and Robotics – A. Bobick
Model Fitting: Hough and RANSAC

Hough transform for circles

- Circle: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- For an unknown radius r, unknown gradient direction

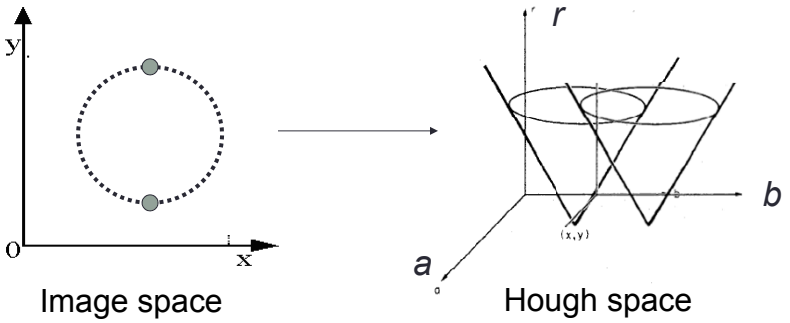


Image space Hough space

Hough transform for circles

- Circle: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- For an unknown radius r, **known** gradient direction

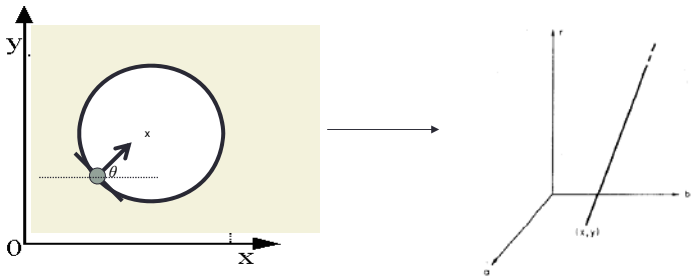


Image space Hough space

Hough transform for circles

For every edge pixel (*x*,*y*) :
  For each possible radius value *r*:
    For each possible gradient direction *θ:*
      %% or use estimated gradient
    a = *x* – *r* cos(*θ*)
    b = *y* + *r* sin(*θ*)
    H[*a*,*b*,*r*] += 1
  end
end

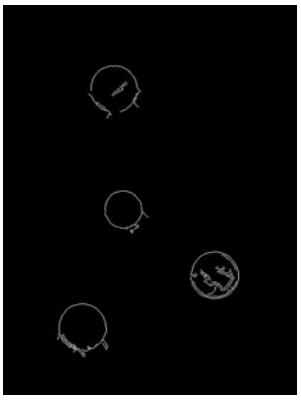
Example: detecting circles with Hough

Crosshair indicates results of Hough transform, bounding box found via motion differencing.

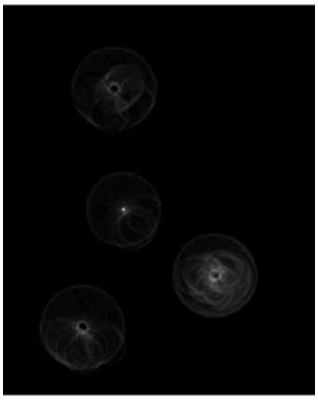
Example: detecting circles with Hough

Original

Edges



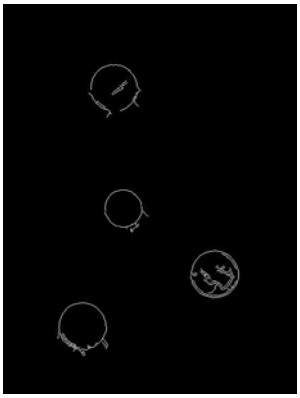
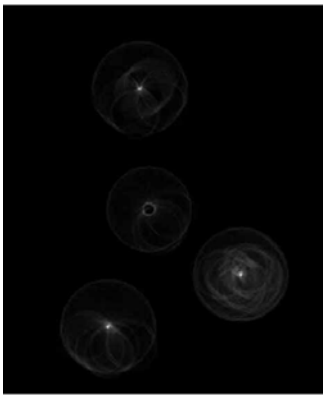
Votes: Penny



Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Example: detecting circles with Hough

Combined detections | Original | Edges | Votes: Quarter

Coin finding sample images from: Vivek Kwatra

Voting: practical tips

- Minimize irrelevant tokens first (take edge points with significant gradient magnitude)
- Choose a good grid / discretization
  - Too coarse: large votes obtained when too many different lines correspond to a single bucket
  - Too fine: miss lines because some points that are not exactly collinear cast votes for different buckets
- Vote for neighbors, also (smoothing in accumulator array)
- Utilize direction of edge to reduce free parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes.

Hough transform: pros and cons

- Pros
  - All points are processed independently, so can cope with occlusion
  - Some robustness to noise: noise points unlikely to contribute consistently to any single bin
  - Can detect multiple instances of a model in a single pass
- Cons
  - Complexity of search time increases exponentially with the number of model parameters
  - Non-target shapes can produce spurious peaks in parameter space
  - Quantization: hard to pick a good grid size
  - ***Complexity of search time increases exponentially with the number of model parameters***

"Find consistent matches"

- Some points (many points) are part of the model we want.
- Some are not
- Need to find the right ones.
- Two well understood approches:
  - Hough Transform – everyone votes for all the models they are comfortable with. Well supported models are chosen.
  - Random Sample Consensus (RANSAC) – a variety of models are proposed until one is found that is supported by a consensus of voters. Discards outliers.

Discard Outliers

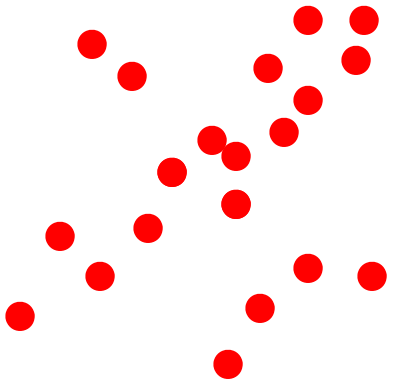
- "No point with d>t"
- RANSAC:
  - RANdom SAmple Consensus
  - Fischler & Bolles 1981
  - Copes with a large proportion of outliers

M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in '81.



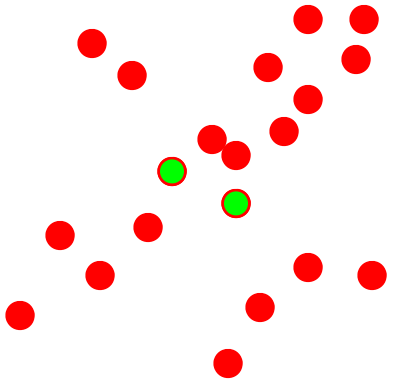
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of ***inliers*** within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



Algorithm:

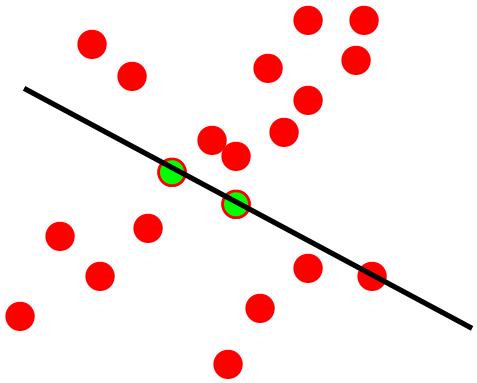
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using sample
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Illustration by Savarese

RANSAC

Line fitting example



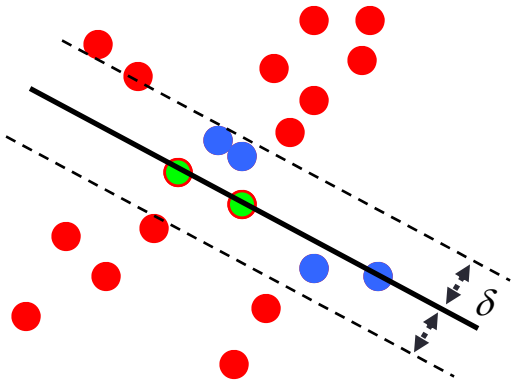
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using sample
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



$N_I = 6$

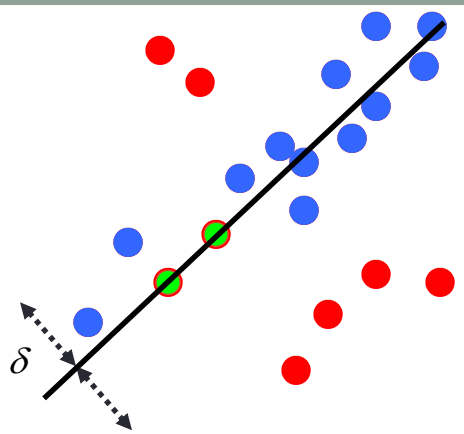
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using the sample
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



$N_I = 14$

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using sample
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Best Line has most support

- More support -> better fit

RANSAC for general model

- A given model has a *minimal set* – the smallest number of samples from which the model can be computed.
  - Line: 2 points
- Image transformations are models. Minimal set of *s* of point pairs/matches:
  - Translation: pick one point pair
  - Homography (for plane) – pick 4 point pairs
  - Fundamental matrix – pick 8 point pairs (really 7 but lets not go there)
- Algorithm
  - Randomly select s points (or point pairs) to form a sample
  - Instantiate a model
  - Get consensus set S_i
  - If $|S_i| > T$, terminate and return model
  - Repeat for N trials, return model with max $|S_i|$

Distance Threshold

- Requires noise distribution
- Location: Gaussian noise with σ
- Distance: *Chi-squared* distribution
  - For 95% cumulative threshold: t=3.84 σ^2
- I.e. -> 95% prob that d<t when point is inlier

How many samples ?

- We want: at least one sample with all inliers
- Can't guarantee: probability p
- E.g. p =0.99

Choosing the parameters

- Initial number of points s
  - Typically minimum number needed to fit the model
- Distance threshold t
  - Choose t so probability for inlier is p (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: t2=3.84σ2
- Number of samples N
  - Choose N so that, with probability p, at least one random sample is free from outliers (e.g. p=0.99) (outlier ratio: e)

Source: M. Pollefeys

Calculate N

- s – number of points to compute solution
- p – probability of success
- *e* – proportion outliers, so % inliers = (1-*e*)
- P(sample set with all inliers)=$(1-e)^s$
- P(sample set will have at least one outlier)= $(1-(1-e)^s)$
- P(all N samples have outlier)=$(1-(1-e)^s)^N$
- We want P(N samples an outlier)<1-p
- $(1-(1-e)^s)^N < 1-p$

$$N > \log(1-p) / \log(1-(1-e)^s)$$

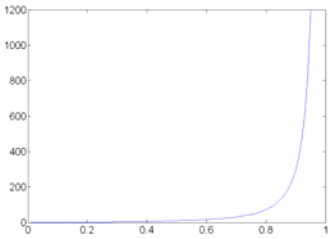
Samples required for inliers only in a sample

- Set p=0.99 – chance of getting good sample
  - s=2, ε=5% => N=2
  - s=2, ε=50% => N=17
  - s=4, ε=5% => N=3
  - s=4, ε=50% => N=72
  - s=8, ε=5% => N=5
  - s=8, ε=50% => N=1177

	proportion of outliers *e*						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

- N = f(ε), *not the number of points*
- N increases steeply with s

$$N > \log(1-p) / \log(1-(1-e)^s)$$



Choosing the parameters

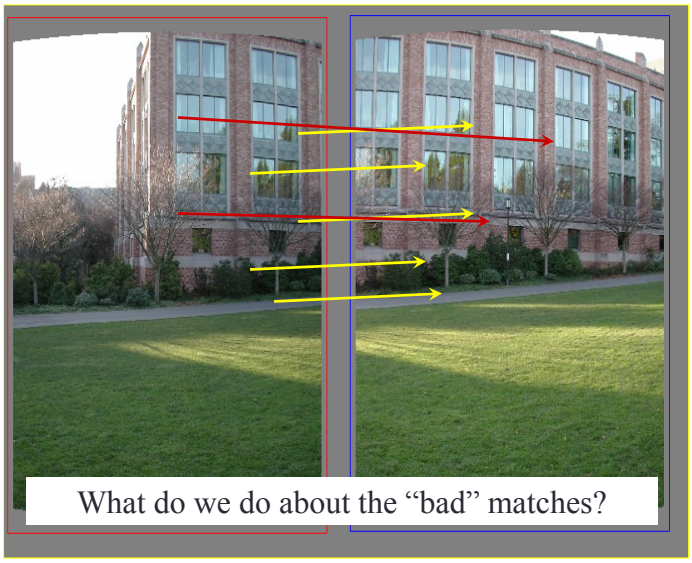
- Initial number of points *s*
  - Typically minimum number needed to fit the model
- Distance threshold *t*
  - Choose *t* so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples *N*
  - Choose *N* so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)

$$N = \log(1-p) / \log(1-(1-e)^s)$$

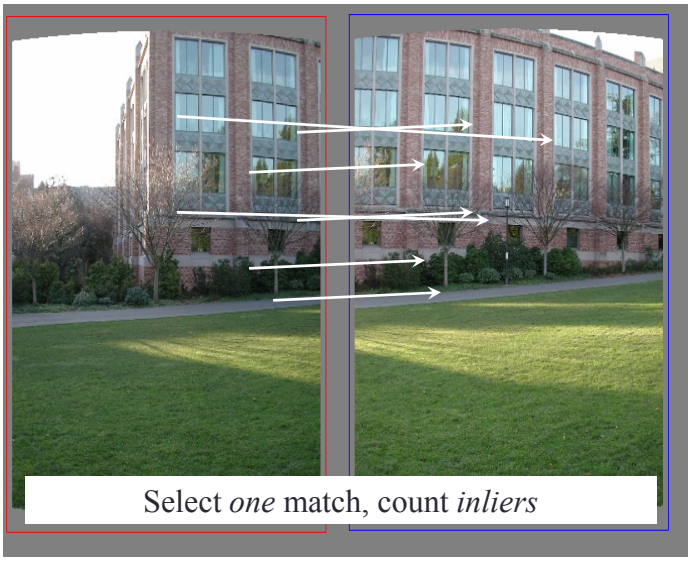
	proportion of outliers *e*						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Source: M. Pollefeys

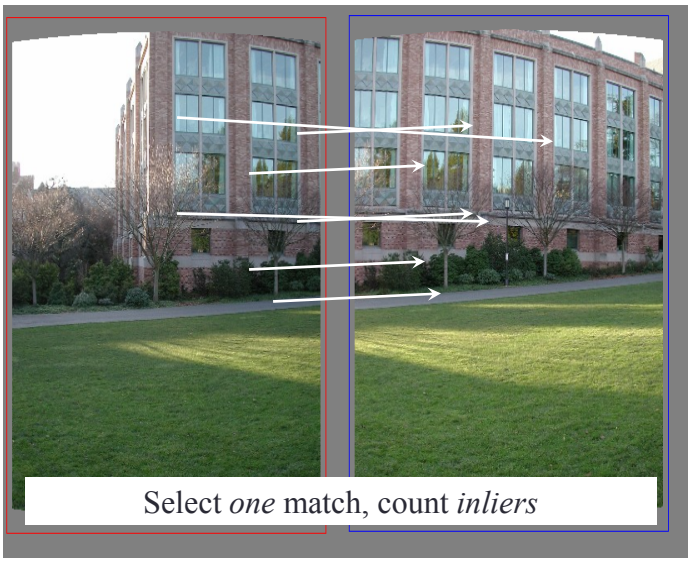
Matching features



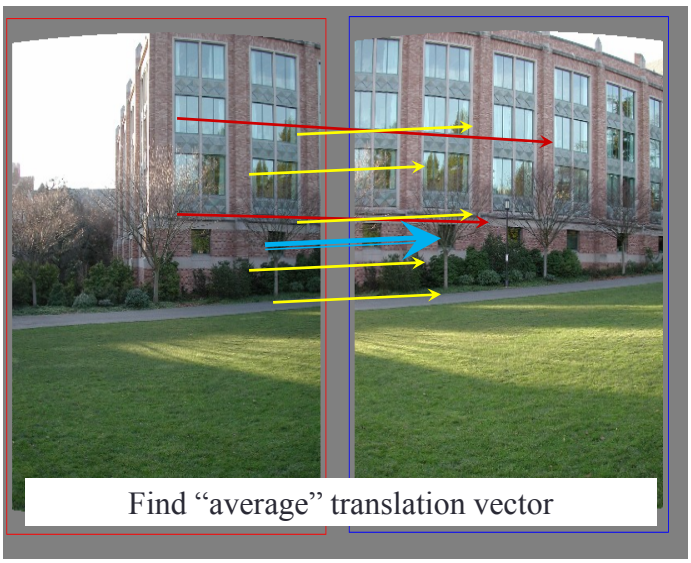
RAndom SAmple Consensus (1)



RAndom SAmple Consensus (2)



Least squares fit



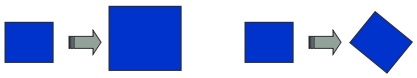
RANSAC for estimating homography

- RANSAC loop:

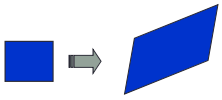
1. Select four feature pairs (at random)
2. Compute homography H (exact)
3. Compute *inliers* where $SSD(p_i', \mathbf{H}\, p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers

2D transformation models

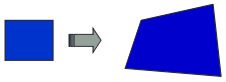
- Similarity (translation, scale, rotation)



- Affine



- Projective (homography)



Source: S. Lazebnik

Adaptively determining the number of samples

- Inlier ratio e is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found, e.g. 80% would yield e=0.2
- Adaptive procedure:
  - N=∞, sample_count =0
  - While N >sample_count
    - Choose a sample and count the number of inliers
    - Set e = 1 – (number of inliers)/(total number of points)
    - Recompute N from e:

$$N = \log(1-p)/\log\left(1-(1-e)^s\right)$$

    - Increment the sample_count by 1

Source: M. Pollefeys

RANSAC conclusions

Good
- Simple and general
- Applicable to many different problems, often works well in practice
- Robust to outliers
- Applicable for larger number of parameters than Hough transform
- Parameters are easier to choose than Hough transform

Bad
- Computational time grows quickly number of parameters
- Not as good for getting multiple fits
- Really not good for approximate models

Common applications
- Computing a homography (e.g., image stitching)
- Estimating fundamental matrix (relating two views)
- Every problem in robot vision: find the table, the floor, the objects on the table…

Generalized Hough transform

- What if want to detect arbitrary shapes defined by boundary points and a reference point?

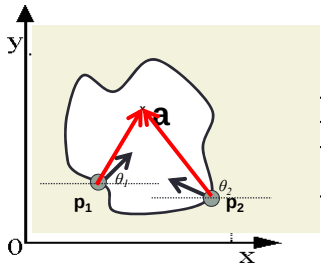


Image space

At each boundary point, compute displacement vector: $\mathbf{r} = \mathbf{a} - \mathbf{p}_i$.

For a given model shape: store these vectors in a table indexed by gradient orientation θ.

[Dana H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980]

The End